Context in Context-Dependent Linear Computation

Abstract

Context-dependent composition of linear operators separates *computation* from *selection*. The previous note considered only the existence of a context function that selects linear operators. This note briefly explores the design space of possible context mechanisms. The objective is not to prescribe a particular routing algorithm but to highlight the surprisingly broad range of information that may define context.

---

1. Context as Computational State

A context function assigns a computational state to an input.

Rather than computing

$$y = Ax,$$

a context-dependent system computes

$$y = A_{c(x)}x,$$

or, for multiple stages,

$$y = A^{(n)}_{c_n} A^{(n-1)}_{c_{n-1}} \cdots A^{(1)}_{c_1} x.$$

The context itself is deliberately left unspecified.

---

2. Sources of Context

Context may be computed from many different sources.

Current Input

The simplest case.

- Binary tests
- Random projections
- Locality-sensitive hashing
- Learned classifiers
- Decision trees
- Another neural network

Previous Inputs

The current computation may depend on recent history.

Examples include

- Sliding windows
- Running averages
- Exponential moving averages
- Short-term memory
- Recurrent state

Internal State

The system may maintain its own computational state.

Examples include

- Finite-state machines
- Learned memory vectors
- Latent variables
- State estimators
- Hidden Markov states

Time

Time itself may provide useful context.

Examples include

- Elapsed time
- Periodic phase
- Iteration count
- Scheduling
- Multi-rate computation

External Signals

Context need not originate from the data being processed.

Examples include

- Robot sensors
- User commands
- Environment state
- Control signals
- System configuration

3. Hysteresis

Context does not necessarily depend only on the present.

Hysteresis allows switching decisions to depend on previous context.

Examples include

- Separate thresholds for entering and leaving a state
- Minimum dwell times
- Transition delays
- Sticky routing
- Confidence margins before switching

Hysteresis may improve stability by preventing rapid oscillation between nearby contexts.

---

4. Hierarchical Context

Context may itself be organized hierarchically.

For example

- coarse task selection,
- intermediate behavioural mode,
- fine computational route.

Each level reduces the search space for the next.

---

5. Soft and Hard Context

Context selection need not be purely discrete.

Possible approaches include

- Hard selection of one operator
- Selection among a small subset
- Weighted combinations of operators
- Probabilistic routing
- Temperature-controlled switching

These choices trade computational efficiency against smoothness during learning.

---

6. Learning Context

The context mechanism may be

- fixed,
- hand-designed,
- partially learned,
- fully learned,
- or adapted online.

Likewise, the operator libraries and context mechanism may be optimized independently or jointly.

---

7. Desirable Properties

Useful context mechanisms may exhibit properties such as

- balanced utilization of operator libraries,
- stability under small perturbations,
- locality of computation,
- computational efficiency,
- compositional reuse,
- robustness to noise,
- temporal consistency,
- interpretable switching behaviour.

The relative importance of these properties will depend on the application.

---

8. Possible Applications

Potential application domains include

- robotics,
- adaptive control,
- reinforcement learning,
- sequence modelling,
- signal processing,
- computer graphics,
- scientific simulation,
- embedded systems,
- sparse machine learning.

The appropriate notion of context is likely to differ substantially between these domains.

---

9. Open Questions

Several research questions naturally arise.

- What makes a good context representation?
- Which context mechanisms are easiest to learn?
- How much memory should context possess?
- When is hysteresis beneficial?
- Should context be local or global?
- How should context be regularized?
- Can useful context hierarchies emerge automatically?
- What is the relationship between context complexity and operator complexity?

---

10. Concluding Remarks

The previous note argued that expressive power can arise from context-dependent composition of linear operators.

This note suggests that the context mechanism itself forms a large and largely independent design space.

The choice of context determines which computations become available, how computation is factorized, and how reusable computational components are combined. Understanding the design of context mechanisms may therefore be as important as understanding the linear operators they select.